UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTAÇÃO E SISTEMAS

KAMILA CRISTINA MOREIRA MARTINS

DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONETES: BOAS PRÁTICAS E DIRETRIZES

João Monlevade
2018

KAMILA CRISTINA MOREIRA MARTINS

DESENVOLVIMENTO BASEADO EM COMPONENTES: BOAS PRÁTICAS E DIRETRIZES

Monografia apresentada ao curso Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina “Trabalho de Conclusão de Curso II”.

Orientador: Diego Zuquim Guimarães Garcia.

João Monlevade

2018

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ATA DE DEFESA

Aos 31 dias do mês de janeiro de 2018, às 18 horas e 00 minutos, na sala C203 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pela aluna **Kamila Cristina Moreira Martins,** sendo a Comissão Examinadora constituída pelos professores: Prof. Dr. Diego Zuquim Guimarães Garcia, Prof. Me. Erik de Britto e Silva e Prof. Me. Theo Silva Lins.

A candidata apresentou a monografia intitulada: *"DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES: BOAS PRÁTICAS E DIRETRIZES"*. A comissão examinadora deliberou, por unanimidade, pela aprovação da candidata, com nota 9,5 (Nove e meio), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final.

Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pela graduanda.

João Monlevade, 31 de janeiro de 2018.

Prof. Dr. Diego Zuquim Guimarães Garcia
Professor Orientador/Presidente

Prof. Me. Erik de Britto e Silva
Professor Convidado

Prof. Me. Theo Silva Lins
Professor Convidado

Kamila Cristina Moreira Martins
Graduanda

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

Curso de Sistemas de Informação

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES: BOAS PRÁTICAS E DIRETRIZES

Kamila Cristina Moreira Martins

Monografia apresentada ao Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CSI499 – Trabalho de Conclusão de Curso II do curso de Bacharelado em Sistemas de Informação e aprovada pela Banca Examinadora abaixo assinada:

Prof. Dr. Diego Zuquim Guimarães Garcia
Departamento de Computação e Sistemas - UFOP

Prof. Me. Erik de Britto e Silva
Departamento de Computação e Sistemas - UFOP

Prof. Me. Theo Silva Lins
Departamento de Computação e Sistemas - UFOP

João Monlevade, 31 de janeiro de 2018

TERMO DE RESPONSABILIDADE

Eu, Kamila Cristina Moreira Martins, declaro que o texto do trabalho de conclusão de curso intitulado — Desenvolvimento de software baseado em componentes: boas práticas e diretrizes, é de minha inteira responsabilidade não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos referidos autores.

João Monlevade, 31 de Janeiro de 2018.

Kamila Cristina Moreira Martins

Kamila Cristina Moreira Martins

Dedico esta, assim como todas as demais conquistas, aos meus amados pais, Danilo e Helena, pelo amor e incentivo incondicional, por serem meus exemplos de vida e por me fortalecerem nos momentos difíceis. “Se cheguei até aqui foi porque me apoiei nos ombros de gigantes”.

AGRADECIMENTOS

Agradeço a Deus, pois sem Ele nada seria possível.

Aos meus pais por acreditarem e serem incentivadores dos meus sonhos.

A minha irmã, Daniela, pelo apoio, companheirismo e amizade.

A tia Dadá e a tia Lica por estarem sempre presentes.

Ao meu namorado, Paulo, pelo carinho, paciência e compreensão inefável.

A todos os amigos e parentes pela contribuição preciosa

Ao professor, Diego, pela orientação, suporte e confiança.

A Universidade Federal de Ouro Preto, ao corpo docente, a direção e administração, pela oportunidade de fazer e concluir o curso nessa Instituição.

A ArcelorMittal pela oportunidade de crescimento pessoal e profissional e a toda a equipe de trabalho.

E não deixando de agradecer de forma grata e grandiosa a todos os envolvidos que fizeram parte da minha formação.

“As alegrias costumam ser preparadas no
silêncio das duras esperas. ”

Padre Fábio de Melo

RESUMO

O atual cenário do mercado de trabalho nos implica a tomar atitudes mais competitivas, no que diz respeito poupar investimentos relacionados a tempo e dinheiro e sempre buscar, cada vez mais, a otimização da qualidade dos produtos e serviços oferecidos ao cliente. Se tratando da Engenharia de Software não seria diferente, uma vez que envolve projetos, processos, recursos, pessoas e claro, resultados. Existem métodos que auxiliam os desenvolvedores de software e o objetivo deste trabalho é apresentar o Desenvolvimento baseado em Componentes (DBC), esse modelo de desenvolvimento se fundamenta na construção de software por meio da junção de componentes já existentes. Esta abordagem acentua o reuso e aponta benefícios no sentido de aumentar produtividade. As vantagens em sua utilização vão desde o desenvolvimento até a manutenção dos sistemas, pois permite atualização, incorporação e/ou substituição dos componentes que já existem.

Este trabalho procura levantar os principais conceitos relacionados ao BDC. A princípio, são apresentadas definições de assuntos relacionados, assim como os componentes, suas características e tipos, modelos de componentes, convenções e framework. Dando destaque a importância da reutilização de software, visando utilizar produtos de software construídos para atender algum problema específico, mas que são utilizados para situações diferentes. Neste contexto se encaixa o DBC, e serão demonstrados alguns de seus efeitos e etapas do processo de desenvolvimento. Por fim, são apresentadas as utilidades e vantagens desse modelo de desenvolvimento que tem sido alvo de estudos por promover uma melhoria considerável no que diz respeito a construção de softwares.

Palavras-chave: Desenvolvimento baseado em componentes. DBC. Software. Reutilização. Reuso.

ABSTRACT

The current labor market scenario makes us to take more competitive actions in order to save investments related to time and money and always seek to optimize the quality of the products and services offered to the customers. Related to Software Engineering it would not be different, since it involves projects, processes, resources, people and of course, results. There are methods that help software developers, and the purpose of this work is to present the Component-based Development (DBC), this development model is based on the development of software through the joining of existing components. This approach accentuates reuse and points to benefits in increasing productivity. The advantages in its use go from the development to the maintenance of the systems, as it allows updating, incorporation and/or replacement of the components that already exist.

This work seeks to raise the main concepts related to DBC. First, definitions of related subjects are presented, as well as the components, their characteristics and types, component models, conventions, and framework. Emphasizing the importance of software reuse, aiming to use software products built to address a specific problem, but which are used for different situations. In this context the DBC fits, and will be demonstrated some of its effects and stages of the development process. Finally, the utilities and advantages of this development model that has been the object of studies for promoting a considerable improvement in the construction of software are presented.

Keywords: Component-based development. DBC. Software. Reuse.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

DBC – Desenvolvimento Baseado em Componentes

SUMÁRIO

1 Introdução

A produtividade e a economia são objetivos que as pessoas e as organizações se esforçam para conseguir alcançar dentro de seus processos de desenvolvimento. Uma vez que, atualmente, o tempo é visto como dinheiro, a eficiência do produto criado é vista como diferencial, e que esses critérios, quando alinhados, agregam confiabilidade por parte do cliente/usuário. Evidentemente, no desenvolvimento de software não seria diferente e, para isso, é necessário o uso de práticas de criação e de gerência que tragam organização e qualidade. Não é uma atividade tão banal definir o desenvolvimento de software, pois a complexidade dos sistemas computacionais tem se tornado cada vez maior e existe uma grande empreitada no sentido de analisar e conseguir alcançar ferramentas ou meios que permitam diminuir as possíveis dificuldades que envolvem recursos pessoais e financeiros investidos na etapa de desenvolvimento.

Existem vários fatores considerados como muito importantes nesse processo, entre eles estão: restrições de negócio (capital e prazo), necessidades e particularidades do cliente e do projeto, técnicas e estratégias que serão utilizados, conformidade com padrões e modelos. Além da necessidade de profissionais altamente qualificados, capazes de articular e combinar todos esses elementos, para conseguir entregar os resultados esperados pela organização.

No intuito de melhorar o desenvolvimento de software elaborou-se um modelo de implementação baseado na reutilização de softwares para criação de componentes. Se trata de reutilizar os códigos, o que não exige um desenvolvimento de linha após linha, reduzindo risco de erros, perda de tempo e minimização de custos. Facilitando mudanças e implantação de novas funções. O Desenvolvimento Baseado em Componentes (DBC) se baseia na reutilização com o conceito de “crie uma vez, use onde quiser”, onde blocos de código já implementados são reutilizados. Por conseguinte, esse desenvolvimento porta de qualidade, flexibilidade e produtividade. Porém, não é tão trivial a utilização do DBC, e uma das maiores dificuldades é a falta de uma padronização ampla e formal para realizá-lo. Em torno disso, será apresentado conceitos, diretrizes, atributos, características e boas práticas para quem optar por esse modelo de desenvolvimento e particularidades e vantagens alcançados a partir da aplicação do DBC.

1.1 Problema

O desenvolvimento de software não é tarefa simples, envolve uma gama de atividades que dependem umas das outras e de pessoas adequadas e com conhecimento suficiente para realiza-las, que no final irão acarretar em um sistema que atenderá aos requisitos do cliente de maneira ligeira e inteligente, afinal, o mercado nos intima a trocar informações de modo eficiente e rapidamente.

Decerto, que o custo do investimento é coroado como o maior dos desafios, pois planejar e elaborar um bom software demanda recursos financeiros que nem sempre estão acessíveis ou ociosos na empresa. Existem sistemas de TI que carecem de suporte 7 dias da semana, 24 horas por dia e disponibilizar esse serviço ininterrupto é dispendioso para as organizações, embora indispensável em casos em que se precisa de reparos para não prejudicar a utilização desses sistemas. Sem contar que a infraestrutura requer alta aplicação, uma vez que abrange servidores, dispositivos de rede e de armazenamento.

Interpretar os requisitos é uma atividade que pode garantir que o sistema seja desenvolvido com as devidas particularidades desejadas pelo cliente. O que não é tão descomplicado quanto parece, os clientes podem não conseguir expressar da melhor maneira possível o que visam alcançar, dificultando a análise dessas exigências. Logo, faz-se necessária a existência de um vínculo contínuo e estreito entre projetista ou desenvolvedor e cliente durante todo o processo de desenvolvimento.

O retrabalho também é um fator desafiador para os desenvolvedores que lidam com a mudança de requisitos por parte dos clientes ou determinações incompletas ou mal explicadas, falta de planejamento, estimativas equivocadas com relação aos prazos que, na maioria das vezes, são apertados, e falta de inspeções no decorrer do desenvolvimento.

O tempo gasto para desenvolver um sistema envolve organização, determinação, disposição e emprego de ferramentas apropriadas. A insatisfação dos clientes se dá, na maioria das vezes, quando os projetos extrapolam os prazos ou são entregues sem que os devidos testes sejam realizados, por falta de tempo. Testes que permitem que a equipe identifique problemas antes que dos usuários, além de garantir segurança do produto.

A boa funcionalidade aliada a usabilidade proporciona uma boa experiência de uso, diminuindo a dificuldade na utilização do software. Resumidamente, um dos diferenciais competitivos dos fornecedores de sistemas de TI é a praticidade e a interação satisfatória de uso disponibilizada aos usuários.

Enfim, a falta de planejamento adequado durante o processo de criação e/ou atualização de um software pode envolver todos os obstáculos citados, gerando quantidade excessiva de tempo e dinheiro destinados a implementação desses sistemas. Esses prejuízos

podem ser evitados por meio de estratégias de reutilização através do DBC que podem contribuir, incrementar e melhorar o desenvolvimento de sistemas computacionais contemplando cada tipo de software.

1.2 Objetivos

O objetivo deste trabalho consiste em realizar um estudo vasto sobre as principais características do Desenvolvimento Baseado em Componentes dentro da Engenharia de Software. Com isso, será feita uma revisão bibliográfica sobre o assunto e serão retratadas as particularidades, características e vantagens alcançadas a partir desse tipo desenvolvimento. Também tem o intuito de assimilar, agregar e praticar os inúmeros conhecimentos obtidos no decorrer do curso.

1.2.1 Objetivo geral

Reunir dicas, diretrizes, boas práticas e orientações para que desenvolvedores que desejam utilizar DBC em seu processo de desenvolvimento de software obtenham sucesso com base nessa escolha.

1.2.2 Objetivos específicos

Os objetivos específicos que irão dar vida ao objetivo geral deste trabalho são:

- Criteriosa e abundante revisão bibliográfica na literatura existente
- Apontar benefícios e vantagens oferecidas pelo DBC
- Sanar dúvidas gerais a respeito dessa implementação

1.3 Justificativa

Dada a importância do desenvolvimento com alta produtividade e com o custo o mais baixo possível, as empresas de tecnologia precisam ir em busca de alguma metodologia que atenda às suas necessidades. O desenvolvimento baseado em componentes pode auxiliar no alcance dos requisitos necessários para se obter sucesso em um software, pois harmoniza a robustez de sistemas já testados com a suas respectivas reutilizações.

1.4 Estrutura do trabalho

Esse trabalho está organizado da seguinte forma:

O capítulo 1 contém a introdução sobre o assunto principal, a problemática envolvida com o tema e o objetivo deste trabalho;

No capítulo 2 dispomos dos conceitos gerais para o entendimento do modelo do DBC e a revisão bibliográfica para embasar o conteúdo abordado;

O capítulo 3 descreve o Desenvolvimento baseado em componentes, suas definições, características, tipos, efeitos do processo, adaptação de componentes e engenharia de Domínio.

O capítulo 4 mostra as conclusões finais, principais aspectos e contribuições obtidas através do trabalho.

E por fim, as referências.

2 Conceitos gerais e revisão da literatura

O atual cenário do mercado, como já dito, nos implica a obter e oferecer respostas rápidas e essa dinâmica pode ser otimizada através do uso de ferramenta de apoio, neste capítulo serão apresentados conceitos relacionados ao DBC que são importantes para o entendimento deste modelo de desenvolvimento, associados ao que existe na literatura bibliográfica que auxiliou e enriqueceu este trabalho.

2.1 Software

“Softwares são os programas de computadores. Se trata de um subsistema de um sistema computacional” como considerou Rezende (2005).

Segundo Pressman (2006), “Um software é composto por instruções de computador, estrutura de dados e documentos. ”

Na maioria das vezes, um software é desenvolvido para resolver algum problema ou otimizar algum processo, porém para que isso de fato aconteça é necessário seguir um conjunto de regras e mecanismos que podem melhorar e incrementar o desenvolvimento.

2.2 Engenharia de Software

“Engenharia de Software é uma disciplina que reúne metodologias, métodos e ferramentas a serem utilizados, desde a percepção do problema até o momento em que sistemas desenvolvido deixa de ser operacional, visando resolver problemas inerentes ao processo de desenvolvimento e ao produto de software”, Carvalho (2001).

Já Macro (1990) definiu engenharia de software como o estabelecimento e uso de bons princípios de engenharia e boas práticas de gerenciamento, além da evolução de ferramentas e métodos para uso apropriado, a fim de obter, dentro dos limites existentes, um software de alta qualidade.

Para Sommerville (1992) a engenharia de software envolve questões técnicas e não técnicas, tais como a especificação do conhecimento, técnicas de projeto e implementação, conhecimentos dos fatores humanos pelo engenheiro de software e ainda, gestão de projetos.

“A criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais” segundo Bauer (1996) apud Pressman (2006).

Segundo Derry (1992), a definição inclui a operação a manutenção do software, não se limitando ao seu desenvolvimento; diz que a abordagem tem que ser sistemática, disciplinada e quantificável e não fala nada de ciência da computação ou matemática. Além disto, a definição inclui o estudo e a busca por abordagens que possibilitem a realização de atividades engenharia de software, isto é, o estudo dos métodos, técnicas, ferramentas, etc. Ele mesmo fez uma reformulação da sua definição: A engenharia de software é aquela engenharia que aplica:

- Uma abordagem sistemática, disciplinada e quantificável;
- Os princípios da Ciência da computação, design, engenharia, administração, matemática, psicologia, sociologia e outras disciplinas se for necessário, e às vezes, pura invenção, para criar, desenvolver, operar e manter de forma econômica, confiável e correta, soluções de alta qualidade para problemas que envolvam software.

Com certeza a engenharia de software pretende garantir vantagens pertinentes derivadas da interação usuário e sistema, dando ênfase a importância das pessoas, que são elementos chave em todo o processo, em busca de respeitar as peculiaridades de cada envolvido. Com isso o processo de software deve ser o melhor possível para que a qualidade seja alcançada e maximizada.

2.3 Processos de software

Processo serve como uma estrutura encadeada de métodos e ferramentas que definem os passos para o desenvolvimento e manutenção do software.

“Um processo de software pode ser definido como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários para conhecer, desenvolver, implantar e manter um produto de software” como definiu Fuggeta (2000).

Conforme Falbo (1998) e Gruhn (2002), “Um processo de software bem definido deve indicar as atividades a serem executadas, os recursos (humanos, de hardware e de software) requeridos, os artefatos consumidos e produzidos e os procedimentos a serem adotados (métodos, técnicas, modelos de documentos, entre outros.

Outra questão que envolve a elaboração de um processo de software é a sua adequação ao domínio de aplicação e ao projeto. A cada projeto, o processo de software deve ser ajustado às especificidades da aplicação, tecnologia a ser utilizada na sua construção, grupo de desenvolvimento e usuários finais (Falbo, 1998), como nenhum projeto é igual ao

outro, para assegurar eficiência é preciso guiar o desenvolvimento a produtos de qualidade, ou seja, garantir uma boa experiência com o software implantado.

Segundo Berger (2003), “O uso de um processo padrão como base para o planejamento do processo de software específico de um projeto permite aos gerentes de projeto definir planos em conformidade com os padrões de qualidade e procedimentos da organização”.

Contudo, ainda é dificultoso comportar todos os tipos de desenvolvimento numa organização, pois cada uma exige caracterização e particularidades de acordo com sua cultura.

2.4 Reutilização de software

Se trata de uma abordagem que parte do princípio de potencializar o uso de softwares já existente, tendo como objetivo: conseguir diminuir custos de produção e manutenção, garantir entregas mais ágeis, tentar agregar mais qualidade e maximizar o retorno sobre o investimento do software. O reuso pode ser do sistema como um todo, isso acontece quando a aplicação toda é reutilizada, ou quando ela é adaptada para atender as particularidades de cada cliente, pode ser de componentes que se trata do uso de subsistemas, ou seja, reuso de média granularidade, como por exemplo: componente arquitetural, ou ainda o reuso de objetos e funções, que é o tipo mais comum e antigo de reutilização e faz referência a componentes que fazem uma única função, por exemplo: funções matemáticas.

Esse modelo de desenvolvimento apresenta, além de muitas vantagens, algumas desvantagens, pois componentes reutilizados podem ficar incompatíveis com versões futuras, falta de ferramentas de suporte pois os ambientes que em que são implantados podem não estar preparados para receber a reutilização e custos que envolvem a procura, entendimento e adaptação do software que se deseja usar. Ou seja, custo e manutenção, falta de ferramentas de suporte e altos investimentos são potenciais problemas, além da “síndrome do não inventado aqui” com a tendência de um bloqueio que existe por parte de algumas organizações que ainda preferem reimplementar algo que já está acessível por não confiar no que já está sendo disponibilizado.

De acordo com Freeman (1968), reutilização é o emprego de alguma informação já acessível que algum desenvolvedor necessite no processo de criação de software.

Segundo Krueger (1992), em âmbito mais geral a reutilização de software pode ser entendida como um processo de criação de software com base em software já existente, ao invés de construí-lo do zero.

Já Sametinger (1997), garante que o reuso pode impactar de forma positiva a qualidade dos produtos de software, englobando custo e produtividade. A reutilização de software resulta em melhorias de qualidade, produtividade, confiabilidade e interoperabilidade. Ainda defende, no que diz respeito a qualidade, que erros podem ser identificados e consertados e que com isso, os componentes reutilizados tendem a possuir mais qualidade, porém esses componentes devem sofrer manutenções para que a melhoria seja alcançada.

Diante dos benefícios apontados por estudos realizados e encontrados na biografia percebe-se que a reutilização de software pode ser vantajosa aos desenvolvedores e clientes, no que diz respeito a recursos investidos durante o desenvolvimento.

2.5 Definição de processos baseada em Reutilização

Vimos a relevância dos processos de software e da reutilização de software. Esses dois conceitos podem ser vistos em uma abordagem que segundo Barreto (2008) “A definição de processos baseados em reutilização tem por objetivo facilitar a definição de processos através de sua composição a partir de componentes de processos reutilizáveis”.

Esta abordagem afirma ainda que é significativo operar um repositório para armazenar todos os componentes de um processo, linhas de processo, conhecimento relacionado à execução dos processos. Com isso, esses repositórios podem ser utilizados durante o desenvolvimento de um projeto, em que componentes ou outros itens que sejam úteis e reutilizáveis possam ser encontrados e utilizados para construir os processos.

2.6 Componentes

Existem na literatura muitas definições de componentes. Segundo Sametinger (1997), “Componentes podem ser vistos como alguma parte do sistema de software que é identificável e reusavel, ou como o estado seguinte de abstração depois de funções, módulos e classes. E ainda que, componentes de software reusáveis são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e deseja-se que tenham interfaces claras, documentação apropriada e uma condição de reuso definida”. A tabela 1 apresenta conceitos dos termos reconhecidos como importantes por Sametinger, para melhor entendimento de sua definição.

Tabela 1 – Conceitos para o entendimento da definição de Componentes

Autocontido	Caraterísticas dos componentes de poderem ser reusáveis sem a necessidade de depender de outros componentes;
Identificação	Componentes devem ser facilmente identificados, ou seja, devem estar contidos em um único lugar ao invés de espalhados e misturados com outros artefatos de software ou documentação. Eles são tratados como artefatos por poderem se tratar de código fonte, documentação ou código executável;
Funcionalidade	Componentes têm uma funcionalidade clara e específica que realizam. Componentes podem realizar funções ou descrever funcionalidades. Com isso se deseja caracterizar toda a documentação do ciclo de vida do software;
Interface	Componentes devem ter uma interface clara, que indica como podem ser reusados e conectados a outros componentes, e devem ocultar os detalhes que não são necessários para o reuso;
Documentação	A existência de documentação é indispensável para o reuso. O tipo de componente e sua complexidade irão indicar a conveniência do tipo de documentação;
Condição de reuso	Componentes devem ser mantidos de modo a preservar o reuso e a condição de reuso compreende diferentes informações como, por exemplo, pessoas envolvidas naquele projeto, proprietário, responsável por algum problema, situação de qualidade, entre outras.

Sametinger (1997) apresentou um conjunto de atributos que ele julgou válidos para o entendimento de componentes:

- Concorrência: Preocupa-se com a concorrência entre componentes e com os aspectos relacionados para que isso se verifique;
- Interatividade: Considera-se diferentes as formas de interação entre componentes e a forma como estes se comportam frente a estas interações;
- Interação: Mostra como acontece a interação entre os componentes e entre os componentes e seus usuários;
- Distribuição: Atributo relacionado aos fatores necessários para a distribuição de componentes permitindo a manutenção de sua comunicação e troca de dados;
- Funcionalidade: Atributo essencial, afinal para que um componente seja reusado ele precisa ser aplicável em determinado contexto;

- Formas de adaptação: Atributo que indica a preocupação em adaptar o componente ao longo do tempo de forma que ele permaneça reutilizável;
- Controle de Qualidade: Preocupação em avaliar e garantir a qualidade do componente.

Essas características quando alinhadas fazem com que o componente seja útil em outros desenvolvimentos, ou seja, que de fato, ele seja eficiente e reusavel, como definiu Sametinger (1997).

De acordo Szyperski (1999), “Componente de software é uma unidade de composição com interfaces contratualmente especificadas e apenas explícitas dependências de contexto. Componentes podem ser utilizados independentes ou combinados com outras partes. É necessário que os componentes especifiquem suas dependências de contexto no que diz respeito à necessidade deles de indicarem o que o ambiente em que serão acionados deve prover para que eles funcionem. ”

Conforme considerou Brown (1997), “Componente é um conjunto independente de serviços reutilizáveis, ou seja, ele provê habilidades que outros componentes podem ter acesso e também que necessitam da existência de uma especificação que apresente o que o componente faz e como se comporta quando os serviços são usados. O que inclusive facilita a substituição de um componente por outro, desde que tenham a mesma especificação. Também significa que a expectativa que um componente coopere para completar uma solução não deve estar vinculada ao contexto para o qual foi criado, dando sentido ao termo independente. ”

Gary e Lindquist (1999) definem um componente como um encapsulamento de informações e comportamentos de processo em um dado nível de subdivisão. E um modelo de processo definido como base em componentes é uma coleção de componentes de processo que interagem de alguma forma. ”

Modelo de componentes

Não se encontra um consenso sobre o que deve estar contido em um modelo de componentes, apenas espera-se que sigam os seguintes padrões e práticas, segundo Bachmann (2002):

Tabela 2 – Convenções esperadas por um modelo de componentes:

Tipos de componente	Formas de interação	Definição de recursos
Definidos em relação as interfaces que são implementadas, sendo que cada interface de um componente corresponde a um tipo. Caso uma interface implemente as interfaces A e B, então ele é do tipo A e B, o que lhe garante uma capacidade polimórfica em relação a estes tipos, ou seja, os componentes assumem diferentes formas de acordo com as funções que exercem. Logo, desempenham papéis e tem formas de interação desiguais .	Se trata de determinar a maneira com que os componentes irão interagir entre si e como irão interagir com o framework de componentes, ou seja, como irão interagir com a biblioteca de classes que suportam uma funcionalidade,	A composição dos componentes se dá pela ligação deles a um ou mais recursos, o modelo de componentes descreve quais recursos estão disponíveis a cada componente e como e quando eles são associados a estes recursos

Com o intuito de complementar a tabela acima, o conjunto de elementos definidos pelo modelo de componentes, descritos por Heineman (2001) é:

- Interface: Especificação do comportamento e propriedades;
- Nomeação: Nomes globais únicos para as interface e componentes;
- Metadados: Informações sobre os componentes, interfaces e seu relacionamento;
- Interoperabilidade: Comunicação e troca de dados entre componentes de diferentes origens e/ou implementados em linguagens distintas;
- Customização: Interfaces que permitem que as interfaces sejam customizadas;
- Composição: Interfaces e regras para combinar ou substituir componentes em uma aplicação;
- Suporte e evolução: Regras que permitem atualizar ou substituir componentes por versões mais novas;
- Empacotamento e utilização: Quando recursos necessários para instalar e configurar determinado componente esteja empacotado as implementações.

Framework

De acordo com Bachmann (2002), framework de componentes representa a base sobre a qual estes padrões e convenções são utilizados. Portanto, modelo de

componente e framework são conceitos complementares e se relacionam fortemente, uma vez que as dimensões estabelecidas devem ser suportadas pelo framework.

Os frameworks são responsáveis por categorias de serviços divididos em distribuição, segurança e gerenciamento de transações e comunicação assíncrona Brown (2000).

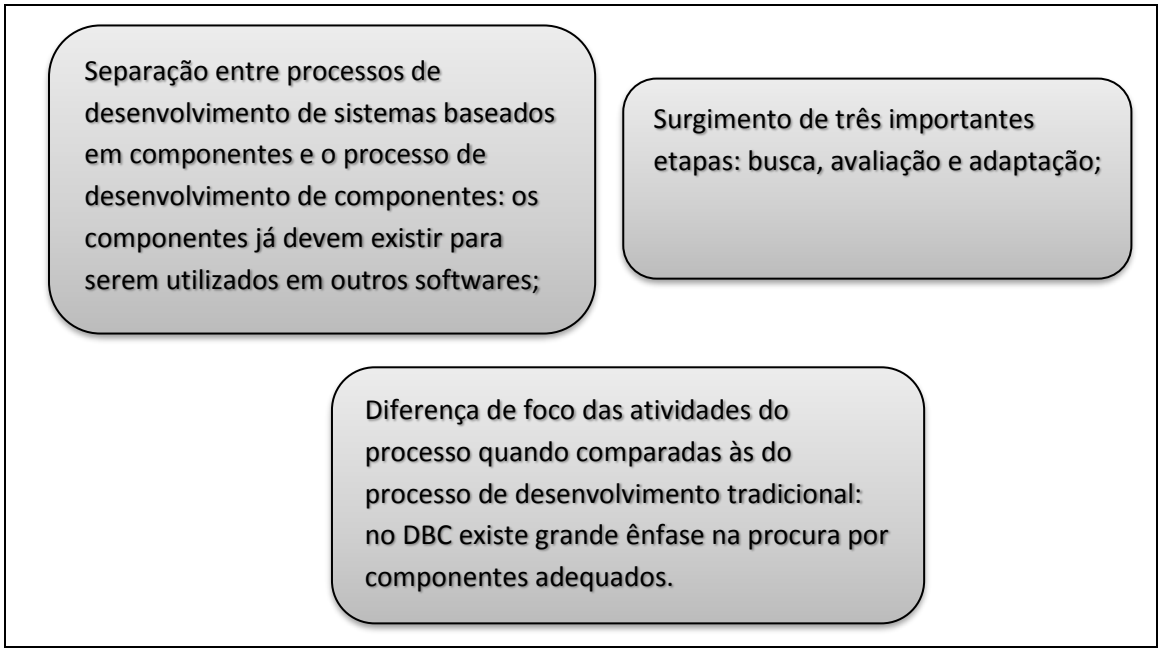
3 Desenvolvimento baseado em componentes - DBC

A preocupação e o trabalho aplicados no intuito de favorecer as técnicas de desenvolvimento de software tentando acrescentar mais qualidade e produtividade, assim como a minimização dos custos esforços, demonstram atuais cenários promissores para o desenvolvimento de sistemas. Nesse contexto, desponta o DBC com uma nova perspectiva de desenvolvimento de software identificada pela composição de fragmentos já existentes.

De acordo com Szyperski (1999), construir novas soluções pela combinação de componentes desenvolvidos e comprados aumenta a qualidade e dá suporte ao rápido desenvolvimento, levando à diminuição do tempo de entrega do produto ao mercado. Os sistemas definidos através da composição de componentes permitem que sejam adicionadas, removidas e substituídas partes do sistema sem a necessidade de sua completa substituição. Com isso, o DBC auxilia na manutenção dos sistemas de software, por permitir que os sistemas sejam atualizados através da integração de novos componentes e atualização dos já existentes.

Conforme Crnkovic (2006), o princípio da abordagem do DBC é construir sistemas com base em componentes já existentes, o que traz alguns efeitos para o processo de desenvolvimento. Algumas delas serão mostradas na Figura 1, a seguir:

Figura 1 – Alguns efeitos do processo de Desenvolvimento Baseado em Componentes



Sametinger (1997) define algumas etapas que devem integrar as atividades do processo de DBC que estão representadas na tabela 2:

Tabela 3 – Etapas do processo de DBC

1º	**Entendimento do Problema**	Estudar o problema e desenvolver uma estrutura de solução baseada em componentes já existentes;
2º	**Reconfiguração**	Adaptar a estrutura de solução para aumentar as chances de se utilizar componentes já disponíveis;
3º	**Busca**	Obter, avaliar e instanciar componentes já existentes;
4º	**Adaptação**	Modificar e adaptar os componentes selecionados que não atendam por completo os requisitos;
5º	**Integração**	Integrar os componentes ao produto final;
6º	**Avaliação**	Avaliar os componentes que precisam ser desenvolvidos e os que foram modificados devido à possibilidade de reutilização e inserção no repositório de componentes

Além disso, Sametinger (1997) afirma que é desejável que um componente tenha independência de hardware, compilador e sistema operacional e ainda possuir documentação adequada.

As atividades essenciais para o DBC, de acordo com Brown (1997), são:

1. *Selecionar* componentes que apresentam potencial de serem usados no desenvolvimento de sistemas, podem ser originários de diferentes fontes, abrangendo os que foram criados para outros projetos e componentes cuja a fonte são diferentes empresas. Portanto, é indispensável investigar as propriedades e qualidade do componente, apesar de que nessa etapa pouco se sabe sobre as características dos componentes. As informações disponíveis podem dizer respeito ao nome do componente, seus parâmetros e uma breve descrição do ambiente de execução. O objetivo de selecionar é gerar uma relação de possíveis componentes, os quais passarão por uma avaliação para saber se realmente serão utilizados. Caso nenhum componente tenha sido selecionado nesta

etapa, passa-se a ter a necessidade de criar um componente do zero, ao invés de reutilizar.

2. *Qualificação* consiste em garantir que o componente candidato execute as funcionalidades necessárias, ajusta-se ao estilo arquitetural definido para o software e apresenta qualidades que são necessárias para aplicação (desempenho, confiabilidade e usabilidade) e ainda de acordo com Brown (1997), são verificadas as interfaces que podem ser conflitantes e identifica-se se há sobreposição entre componentes. A qualidade geralmente envolve uma análise detalhada de toda a documentação ou especificação disponível, discussão com os desenvolvedores dos componentes e usuários, e teste de execução do componente em diferentes cenários. Quanto mais rigorosas forem as descrições dos componentes disponíveis, mais a atividade de qualificação pode ser reduzida e validar o funcionamento através das experiências e do uso temporário do componente.
3. *Adaptação* corrige potencias fontes de conflito entre os componentes selecionados e qualificados para compor o sistema, na maioria das vezes o reuso de um componente está relacionado a alguma forma de adaptação para que ele atenda aos requisitos.
4. *Composição* é realizada através de uma infraestrutura que provem a ligação dos componentes e um conjunto de convenções conceituais compartilhadas pelos componentes.
5. *Atualização* dos componentes, onde versões antigas serão substituídas ou novos componentes serão incluídos. Essa atividade tem importantes implicações, pois sem planejamento, uma alteração em um componente pode provocar inúmeras repercussões inesperadas em muitos outros componentes do sistema.

Apesar de não existir uma sequência correta de atividades ou a obrigação de obedece-las, Brown (1997) descreve estas como sendo as proáveis atividades realizadas no DBC.

Embora identificados muitos benefícios diante dessa abordagem, segundo Jaufman e Munch (2005), a principal deficiência da definição de processos baseada em componentes é a falta de apoio à adaptação dos processos, sendo sua principal vantagem o fato de permitir que apenas parte do processo sejam reutilizadas. Por outro lado, a abordagem

de adaptação de processos genéricos tem como principal vantagem o apoio à adaptação dos processos, tornando-os mais consistentes. Apesar de que essas abordagens não tratam a reutilização de fragmentos de processos. A adaptação de componentes é uma atividade que será apresentada com mais detalhes, pois sua importância surge a partir do momento que os requisitos não são iguais, uns aos outros, e que cada cliente terá suas particularidades e exigências.

3.1 Adaptação de componentes

Visando desenvolver métodos que simplifiquem a definição dos processos genéricos e a adaptação desses processos para os projetos, levando em consideração suas particularidades, alguns tópicos viraram alvo de estudos.

De acordo com Reis (2002), utilizar processos genéricos para definição dos processos de projeto é uma alternativa para facilitar a tarefa. No entanto, para que isso seja possível, é necessário que haja mecanismos de definição dos processos e de adaptação dessas soluções para os projetos específicos.

Numa definição simples e curta levantada por Wang e King (2000), os processos genéricos fornecem modelos de soluções para problemas comuns em um nível elevado de abstração.

3.2 Engenharia de Domínio

Com o intuito de agrupar um conjunto de sistemas ou de áreas funcionais, que demonstrem funcionalidades similares, surgiu a Engenharia de Domínio, que dá base para a reutilização de componentes na abordagem do DBC.

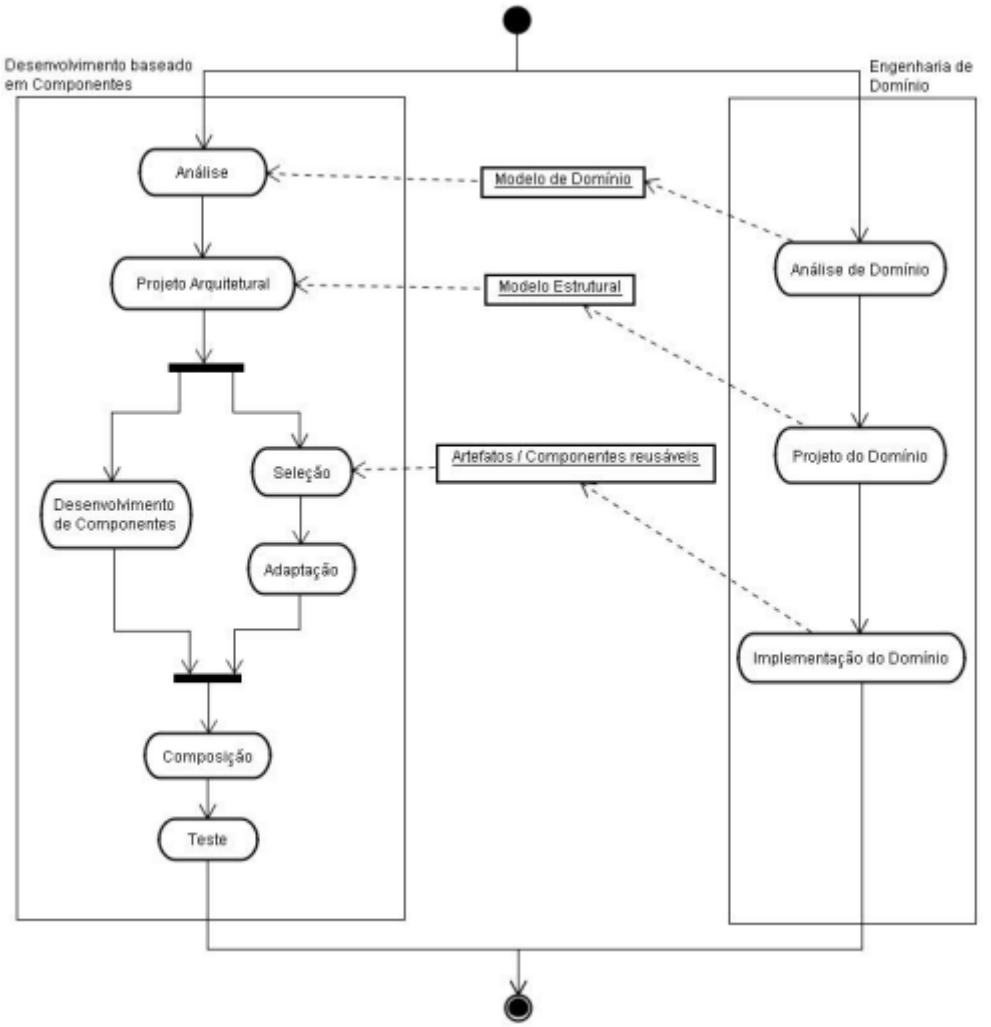
“No entanto é muito difícil criar um componente reutilizável, como alternativa para resolver esse problema, surge a Engenharia de Domínio, auxiliando no desenvolvimento de componentes reutilizáveis em um dado domínio”, Gimenes e Huzita (2005).

A Engenharia de Domínio tem como objetivo, construir componentes reutilizáveis que solucionem problemas de domínios de conhecimento específicos, de acordo com Werner e Braga (2000).

Um processo de desenvolvimento baseado em componentes deve conter caminhos paralelos, que permitam ocorrer, simultaneamente, a engenharia de domínio e o DBC. A engenharia de domínio busca estabelecer componentes de software que possam ser reusados ao longo de todas as etapas do DBC. É primordial que um componente utilizado na

fase de especificação do sistema seja consistente e compatível com os utilizados nas etapas de projeto e implementação, por exemplo, conforme definiu Pressman (2006). A figura 2 a seguir, mostra um modelo que abrange as duas abordagens.

Figura 2 – Modelo de Processo de Desenvolvimento Baseado em Componentes (adaptada de PRESSMAN, 2006)



5 Conclusões

As organizações que se preocupam com a qualidade, organização e eficácia do seu processo de software aliadas ao baixo investimento e produtos e serviços entregues dentro do prazo, precisam captar a importância de abordagens que auxiliem o desenvolvimento de seus projetos. Esses métodos estão intimamente ligados a qualidade do processo como um todo.

Nesse contexto, este trabalho teve o intuito de apresentar o DBC, suas definições e características, retratar a importância da reutilização de software, dos componentes e seus respectivos padrões e convenções, das adaptações para atender as particularidades e garantir flexibilidade a cada componente e do trabalho paralelo entre Engenharia de Domínio e o DBC afim de assegurar agilidade, uma vez que ela permite agrupar componentes de acordo com suas funcionalidades.

O procedimento de pesquisa usado neste trabalho foi unicamente a revisão bibliográfica, no qual procurou-se consolidar os fundamentais conceitos referentes as principais definições, características e vantagens do processo de DBC. Foi identificada grande convergência na literatura, uma vez que este assunto vem sendo alvo de pesquisa, devido ao atual cenário do mercado de trabalho.

Como contribuição deste estudo, procurou-se categorizar as particularidades e propriedades desse modelo de desenvolvimento de software baseado em componentes, como maneira de orientar as pessoas que necessitem de informações pertinentes a este cenário.

Posteriormente, na possível continuidade desse processo de pesquisa pretende-se progredir o estudo tendendo ao detalhamento dos frameworks de desenvolvimento baseado em componentes existentes.

Referências

BACHMANN, Félix et al. Volume II: Technical Concepts of Component-Based Software Engineering – 2Edition.

BARRETO, Ahilton. Componentizando processos legados de software visando a reutilização de processos. Ouro Preto 2009.

BAUER, C. Agregando frameworks de infra-estrutura em uma arquitetura baseada em componentes. Rio de Janeito, 1996.

BROWN, Alan W., On Components and Objects: The Foundation of Components-Based Development. Pittsburgh, PA: IEEE Press, 1997.

BROW, Alan W. Wallanau, Kurt C., The Current State of CBSE IEE Software, 1998.

BROW, Alan W., Large-scale component-based development. Upper Saddle River, 2000.

FALBO, R. Construção de Ambientes de Desenvolvimento de Software. Rio de Janeiro, 1998.

FUGGETA, Alfonso. Software Process: A roadmap. ACM Press, 2000.

GIMENES, Itana M. de S., HUZITA, Elisa H. M., Desenvolvimento baseado em componentes: conceitos e técnicas. Ciência Moderna, 2005.

GUERRA, Paulo A. D., MORONTE, Tiago, TOMITA, Rodrigo and RUBIRA Cecília. Um modelo Conceitual para o desenvolvimento baseado em componentes e centrado na arquitetura de software, 2005.

HEINEMAN, George T., An Evaluation of Component Adaptation Techiques. In: Internation Workshop on Componet-based software Engineering, 1999.

HEINEMAN, George T., Component-Based Software Engineering: putting the pieces together. New York, 2001.

KRUEGER, Lawrence F. Construção e Reutilização de Componentes de Software, UFSC, 1992.

MACRO, A. Software Engineering – Concepts and Managements. Prentice Hall, 1990.

JAUFMAN, O., MUNCH, J. Acquisition of a Project-Specific Process". Finland, 2005.

PRESSMAN, Roger S. Software Engineering: a practioner's approach. New York, 2001

REIS, Gustavo M., Comparações de testes paramétricos e não paramétricos aplicados em delineamentos experimentais. UFV, 2002.

SAMETINGER, Johannes. Software Engineering with Reusable Components. New York, 1997.

SZYPERSKI, Clements. Component Software: Beyond objetc-oriented programming. Harlow, 1997.

WERNER, Claudia M. L., BRAGA, Regina M. M. Desenvolvimento Baseado em Componentes. UFRJ, 2000.